DEPENDABLE SYSTEMS AND SOFTWARE

Fachrichtung 6.2 — Informatik
Prof. Dr.-Ing. Holger Hermanns
Dipl.-Inform. Lijun Zhang

Übungsblatt 4 (Programmierung I)

Lesen Sie im Skript: Kapitel 3.6-4.3

Aufgabe 4.1: (Lexikalische Bindung)
Betrachten Sie den Ausdruck

```
(fn x => (fn y => (fn x => y) x) y) x
```

Geben Sie die Baumdarstellung des Ausdrucks an und stellen Sie die lexikalischen Bindungen durch Pfeile dar. Bereinigen Sie den Ausdruck durch Indizieren der benutzenden Bezeichnerauftreten und überstreichen der definierenden Bezeichnerauftreten. Geben Sie alle Bezeichner an, die in dem Ausdruck frei auftreten.

Aufgabe 4.2: (Bereinigung)
Betrachten Sie das Programm

```
val (x, y) = (2, 3)
fun f x x = #1(x, y)
val y = x*y
fun g g = f x g
```

Klären Sie zuerst mithilfe eines Interpreters, wie die Deklarationen der Prozeduren f und g zu verstehen sind. Bereinigen Sie dann das Programm durch Indizieren der benutzenden Bezeichnerauftreten und Überstreichen der defnierenden Bezeichnerauftreten.

Aufgabe 4.3: (Bereinigung und semantisch äquivalente Ausdrücke)
Betrachten Sie die Deklaration

```
fun f (x, y) = (fn x => (fn y => (fn x => y) x) y) x
```

(a) Stellen Sie die lexikalischen Bindungen der Deklaration durch Pfeile dar.

(b) Bereinigen Sie die Deklaration durch Indizieren der benutzenden Bezeichnerauftreten. Überstreichen Sie die definierende Bezeichnerauftreten. Ist die Deklaration offen oder geschlossen?

(c) Geben Sie das Typschema an, mit dem der deklarierte Bezeichner f getypt wird. Geben Sie dazu passend die Typen der definierenden Auftreten der Bezeichner x und y an.

(d) Geben Sie eine möglichst einfache semantisch äquivalente Deklaration an, die ohne Abstraktionen gebildet ist.

Aufgabe 4.4: (Exists)
Deklarieren Sie eine Prozedur

$$exists : int \to int \to (int \to bool) \to bool$$

die testet, ob eine Prozedur für mindestens eine Zahl zwischen Zahlen den Wert *true* liefert. Orientieren Sie sich an der Prozedur $forall$ aus §3.7.

Aufgabe 4.5: (Andalso als Prozedur)
Warum können Sie einen Ausdruck e_1 *andalso* e_2 nicht immer durch eine Anwendung der Prozedur

```
fun andalso' x y = if x then y else false
```

ersetzen? Hinweis: Lösen Sie zuerst Aufgabe 3.15.

Aufgabe 4.6: (Primzahlen mit Hilfsprozeduren)
Deklarieren Sie semantisch äquivalente Prozeduren zu *sqrt*, *prime*, *nextprime* und *nthprime* (siehe §3.8), die ohne Rückgriff auf höherstufige Prozeduren formuliert sind. Verwenden Sie dabei passende Hilfsprozeduren.

Aufgabe 4.7: (Reduce)
Schreiben Sie eine Prozedur $reduce : int \to int \to int * int$, die zu zwei Zahlen $n, p \geq 2$ das eindeutig bestimmte Paar (m, k) liefert, sodass $n = m \cdot p^k$ gilt und m nicht durch p teilbar ist.

Aufgabe 4.8: (Listendarstellungen)
Geben Sie einen Ausdruck an, der die Liste [7,2,4,1] mit Cons und Nil beschreibt. Geben Sie die Baumdarstellung dieser Liste an.

Aufgabe 4.9: (Baumdarstellung von Listen)
Betrachten Sie den Ausdruck $1 :: 2 :: 3 :: nil@4 :: 5 :: nil$.
a) Geben Sie die Baumdarstellung des Ausdrucks an. Beachten Sie dabei die Klammersparregeln fr :: und @.
b) Geben Sie die Baumdarstellung der durch den Ausdruck beschriebenen Liste an.

Aufgabe 4.10: (Klammersparregeln)
Macht es für die dargestellten Listen einen Unterschied, wie die folgenden Ausdrücke geklammert sind?

(a) `(e1::e2)@e3 oder e1::(e2@e3).`

(b) `(e1@e2)@e3 oder e1@(e2@e3).`

(c) `(e1::e2)::e3 oder e1::(e2::e3).`

Aufgabe 4.11: (Enum)
Schreiben Sie eine Prozedur enum: $int * int \to int\ list$, die zu zwei Zahlen $m \leq n$ die liste [m,...,n] liefert. Beispielsweise soll enum(3, 6) = [3,4,5,6] gelten. Für $m > n$ soll enum die leere Liste liefern.

Aufgabe 4.12: (tab)
Schreiben Sie mithilfe der Prozedur *iter* aus §3.9 eine Prozedur $tab : int \to (int \to \alpha) \to \alpha\ list$, die für n und f dasselbe Ergebnis liefert wie *tabulate* für (n,f).

Aufgabe 4.13: (member)
Schreiben Sie eine polymorphe Prozedur

$$member :' \alpha \to' \alpha\ list \to bool$$

die testet, ob ein Wert als Element in einer Liste vorkommt.

Aufgabe 4.14: (Listendarstellung der natrlichen Zahlen)
Wir wollen die Listen über *unit* als Darstellungen der natürlichen Zahlen auffassen. Dabei soll eine Liste der Länge n die natürliche Zahl n darstellen. Beispielsweise soll die Liste $[(), (), ()]$ die Zahl 3 darstellen. Da *unit* nur einen Wert hat, ist die Darstellung eindeutig.

a) Deklarieren Sie eine Prozedur $rep : int \to unit\ list$, die die Listendarstellung einer natürlichen Zahl liefert.

b) Deklarieren Sie eine Prozedur $num : unit\ list \to int$, die zu einer Liste die dargestellte Zahl liefert.

c) Deklarieren Sie für die Listendarstellungen kaskadierte Prozeduren *add*, *mul* und *less*, die den Operationen $+$, $*$ und $<$ für natürliche Zahlen entsprechen. Verwenden Sie dabei keine Operationen für *int*.

Aufgabe 4.15: (ambige Deklaration)
Wie typisiert die ambige Deklaration $val\ f = map\ rev$ den Bezeichner f? Wie müssen Sie die Deklaration umschreiben, damit f an eine polymorphe Prozedur gebunden wird?

Aufgabe 4.16: (Member)
Schreiben Sie mithilfe von *List.exists* eine Prozedur $member : ''a \to ''a\ list \to bool$, die testet, ob ein Wert als Element in einer Liste vorkommt.